CALM WAVE 포팅 메뉴얼

목차

I. 빌드 및 배포

II. 외부 서비스

I. 빌드 및 배포

1. 개발 환경

Server : AWS EC2 Ubuntu 22.04 LTS
Visual Studio Code : 1.75.0
IntelliJIDEA : 2022.3.1(Ultimate Edition)
JVM : openjdk 11.0.17 2022-10-18
Docker : 20.10.23
Node.js : 18.13.0 LTS
MySQL : 8.0.32 for Linux on x86_64
Redis : 7.0.8
Nginx : 1.18.0 (Ubuntu)
Jenkins : 2.375.2
Openvidu : 2.25.0
Teachable Machine : 2.4.5

2. 설정 파일 정보

React

- Dockerfile

```
FROM node:alpine
WORKDIR /usr/src/app
COPY ./package* /usr/src/app/
RUN npm install
COPY ./ /usr/src/app/
CMD ["npm","run","start"]
```

Spring

- Dockerfile

```
FROM openjdk:11-jdk
ARG JAR_FILE=build/libs/*.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java","-jar","-Duser.timezone=Asia/Seoul", "-Dspring.profiles.active=prod",
"/app.jar"]
```

- application.yml

```
spring:
  config:
    activate:
      on-profile: local
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url:
jdbc:mysql://localhost:3306/calmwave?serverTimezone=Asia/Seoul&useUnicode=true&characterEncoding=utf8
    username: [로컬 디비 username]
    password: [로컬 디비 pw]
  mvc:
    pathmatch:
      matching-strategy: ant_path_matcher
  redis:
    lettuce:
      pool:
        max-active: 5
        max-idle: 5
        min-idle: 2
    host: localhost
    port: [레디스 포트]
  jpa:
    hibernate:
      ddl-auto: update
    properties:
      hibernate:
        format_sql: true
        open-in-view: true
  logging.level:
```

```
    org.hibernate.SQL: debug
my:
  secret: ${JWT_SECRET_KEY}
server:
  port: 8080
  servlet:
    context-path: /
    encoding:
      charset: UTF-8
      enabled: true
      force: true
---
spring:
  config:
    activate:
      on-profile: prod
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: ${SPRING_MYSQL_URL}
    username: ${SPRING_MYSQL_USERNAME}
    password: ${SPRING_MYSQL_PASSWORD}
  mvc:
    pathmatch:
      matching-strategy: ant_path_matcher
  redis:
    lettuce:
      pool:
        max-active: 5
        max-idle: 5
        min-idle: 2
```

```
    host: [도메인]
    port: [레디스 포트]
  jpa:
    hibernate:
      ddl-auto: update
    properties:
      hibernate:
        format_sql: true
    open-in-view: true
  logging.level:
    org.hibernate.SQL: debug
my:
  secret: ${JWT_SECRET_KEY}
server:
  port: 8080
  servlet:
    context-path: /
    encoding:
      charset: UTF-8
      enabled: true
      force: true
```

3. 빌드 서버에 Docker, 자바, 깃, 젠킨스 설치

- Docker 인스톨

```
sudo apt-get update

sudo apt-get install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release
```

```
sudo mkdir -p /etc/apt/keyrings

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt-get update

sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin
```

- Docker Compose 인스톨

```
sudo curl -L \
"https://github.com/docker/compose/releases/download/1.28.5/dockercompose-$(uname -s)-$(uname -m)" \
-o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose
```

- 자바 설치

```
sudo apt-get update
sudo apt-get install openjdk-11-jdk
```

- 깃 설치

```
sudo apt-get install -y git
```

- 젠킨스 설치 (docker-compose.yml)

```
version: "3"
services:
  jenkins:
    privileged: true
    restart: always
    container_name: jenkins
```

```
    image: jenkins/jenkins:lts
    user: root
    ports:
      - "[설정할 호스트 포트]:8080"
      - "50010:50000"
    expose:
        - "8080"
        - "50000"
    volumes:
        - './jenkins:/var/jenkins_home'
        - '/var/run/docker.sock:/var/run/docker.sock'
    environment:
        TZ: "Asia/Seoul"
```

- 젠킨스 컨테이너 안에 docker-cli 설치

4. 배포 서버에 Docker, 자바, mysql, redis 설치

- Docker 설치 (같으므로 생략)

- 자바 설치 (같으므로 생략)

- mysql, redis 설치 (docker-compose.yml)

```
version: '3'
services:
  redis:
    restart: always
    image: redis
    container_name: redis
    ports:
      - [포트번호]:6379
  mysql:
    restart: always
    image: mysql
    container_name: mysql
    ports:
      - [포트번호]:3306
```

```
    volumes:
      - /mysql:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: "${DB_ROOT_PASSWORD}"
      MYSQL_DATABASE: "${DB_DATABASE}"
      MYSQL_USER: "${DB_USER}"
      MYSQL_PASSWORD: "${DB_PASSWORD}"
      TZ: Asia/Seoul
    command:
      - --character-set-server=utf8mb4
      - --collation-server=utf8mb4_unicode_ci
```

5. 배포 서버에 nginx 설치 및 https 적용

- nginx 설치, Let's Encrypt 설치 및 SSL 인증서 발급

```
sudo apt-get install nginx

sudo systemctl stop nginx

sudo apt-get install letsencrypt

sudo letsencrypt certonly –standalone –d [도메인]
```

- nginx.conf

```
server {
        listen 80;
        server_name [도메인];
        server_tokens off;
    if ($host = [도메인]) {
                return 301 https://$host$request_uri;
    } # managed by Certbot
    return 404; # managed by Certbot
}
server {
        listen 443 ssl;
        server_name [도메인];
        server_tokens off;
```

```
        ssl_certificate /etc/letsencrypt/live/[도메인]/fullchain.pem;
        ssl_certificate_key /etc/letsencrypt/live/[도메인]/privkey.pem;
        location /{
                proxy_pass http://localhost:[프론트 포트];
        }
        location /api/ {
                proxy_pass http://localhost:[백엔드 포트]/;
        }
}
```

- 파일 연동 및 테스트

```
sudo ln -s /etc/nginx/sites-available/nginx.conf /etc/nginx/sites-enabled/nginx.conf
sudo nginx -t
```

- nginx 재시작

```
sudo systemctl restart nginx
```

6. Jenkins 사용한 빌드 배포 자동화

- 젠킨스 플러그인 설치

Gitlab Plugin, Generic Webhook Trigger Plugin, docker pipeline plugin, ssh agent plugin 설치

- 백엔드 파이프라인

```
pipeline {
    agent any
    environment {
        repository = "[도커허브 리포지토리]"
    }
    stages {
        stage("Set Variable") {
            steps {
                script {
                    PREV_BUILD_NUM = ("${env.BUILD_NUMBER}" as int) - 1
```

```
                CONTAINER_NAME = "backend"
                DOCKER_HUB_CREDENTIAL = "dockerhub-credential"
                SSH_CONNECTION = "ubuntu@[도메인]"
                SSH_CONNECTION_CREDENTIAL = "Deploy-Server-SSH-Credential"
                GIT_CRED = "git-credential"
                GIT_URL = "[깃 url]"
            }
        }
    }

    stage('checkout') {
        steps {
            git branch: 'release/BE',
                credentialsId: "${GIT_CRED}",
                url: "${GIT_URL}",
                poll: true,
                changelog: true
        }
    }

    stage("Clean Build Test") {
        steps {
            dir ('backend') {
                sh 'chmod +x gradlew'
                sh './gradlew clean bootJar'
            }
        }
    }

    stage("Build Container Image") {
        steps {
            script {
                dir ('backend') {
                    image = docker.build repository + ":backend_${env.BUILD_NUMBER}"
                }
            }
        }
    }
```

```
        stage("Push Container Image To Docker Hub") {
            steps {
                script {
                    docker.withRegistry("https://registry.hub.docker.com",
DOCKER_HUB_CREDENTIAL) {
                        image.push("backend_${env.BUILD_NUMBER}")
                    }
                }
            }
        }

        stage("Cleaning up") {
                    steps {
                sh """docker images | grep "backend_" | awk '{print \$1 ":" \$2}' | xargs docker rmi"""
// docker image 제거
            }
        }

        stage("Server Run") {
            steps {
                sshagent([SSH_CONNECTION_CREDENTIAL]) {
                    // 이전 컨테이너 삭제
                    sh "ssh -o StrictHostKeyChecking=no ${SSH_CONNECTION} 'docker rm -f
${CONTAINER_NAME}'"
                    // 이전 이미지 삭제
                    sh "ssh -o StrictHostKeyChecking=no ${SSH_CONNECTION} 'docker rmi -f
$repository:backend_${PREV_BUILD_NUM}'"
                    // 최신 이미지 PULL
                    sh "ssh -o StrictHostKeyChecking=no ${SSH_CONNECTION} 'docker pull
$repository:backend_${env.BUILD_NUMBER}'"
                    // 이미지 확인
                    sh "ssh -o StrictHostKeyChecking=no ${SSH_CONNECTION} 'docker images'"
                    // 환경변수 파일 실행권한 주기
                    sh "ssh -o StrictHostKeyChecking=no ${SSH_CONNECTION} 'chmod +x
/home/ubuntu/main.env'"
                    // 최신 이미지 RUN
                    sh "ssh -o StrictHostKeyChecking=no ${SSH_CONNECTION} 'docker run -d --
name ${CONTAINER_NAME} --env-file /home/ubuntu/main.env -p [포트번호]:8080
$repository:backend_${env.BUILD_NUMBER}'"
```

```
                    // 컨테이너 확인
                    sh "ssh -o StrictHostKeyChecking=no ${SSH_CONNECTION} 'docker ps'"
                }
            }
        }
    }
}
```

- 프론트엔드 파이프라인

```
pipeline {
    agent any
    environment {
        repository = "[도커허브 리포지토리]"
    }
    stages {
        stage("Set Variable") {
            steps {
                script {
                    PREV_BUILD_NUM = ("${env.BUILD_NUMBER}" as int) - 1
                    CONTAINER_NAME = "frontend"
                    DOCKER_HUB_CREDENTIAL = "dockerhub-credential"
                    SSH_CONNECTION = "ubuntu@[도메인]"
                    SSH_CONNECTION_CREDENTIAL = "Deploy-Server-SSH-Credential"
                    GIT_CRED = "git-credential"
                    GIT_URL = "[깃 url]"
                }
            }
        }

        stage('checkout') {
            steps {
                git branch: 'release/FE',
                    credentialsId: "${GIT_CRED}",
                    url: "${GIT_URL}",
                    poll: true,
                    changelog: true
            }
```

```
        }

        stage("Build Container Image") {
            steps {
                script {
                    dir ('frontend') {
                        image = docker.build repository + ":frontend_${env.BUILD_NUMBER}"

                    }
                }
            }
        }

        stage("Push Container Image To Docker Hub") {
            steps {
                script {
                    docker.withRegistry("https://registry.hub.docker.com",
DOCKER_HUB_CREDENTIAL) {
                        image.push("frontend_${env.BUILD_NUMBER}")
                    }
                }
            }
        }

        stage("Cleaning up") {
                    steps {
                sh """docker images | grep "frontend_" | awk '{print \$1 ":" \$2}' | xargs docker rmi"""
// docker image 제거
            }
        }

        stage("Server Run") {
            steps {
                sshagent([SSH_CONNECTION_CREDENTIAL]) {
                    // 이전 컨테이너 삭제
                    sh "ssh -o StrictHostKeyChecking=no ${SSH_CONNECTION} 'docker rm -f
${CONTAINER_NAME}'"
                    // 이전 이미지 삭제
                    sh "ssh -o StrictHostKeyChecking=no ${SSH_CONNECTION} 'docker rmi -f
```

```
$repository:frontend_${PREV_BUILD_NUM}'"
                    // 최신 이미지 PULL
                    sh "ssh -o StrictHostKeyChecking=no ${SSH_CONNECTION} 'docker pull $repository:frontend_${env.BUILD_NUMBER}'"
                    // 이미지 확인
                    sh "ssh -o StrictHostKeyChecking=no ${SSH_CONNECTION} 'docker images'"
                    // 최신 이미지 RUN
                    sh "ssh -o StrictHostKeyChecking=no ${SSH_CONNECTION} 'docker run -d --name ${CONTAINER_NAME} -p [포트번호]:3000 $repository:frontend_${env.BUILD_NUMBER}'"
                    // 컨테이너 확인
                    sh "ssh -o StrictHostKeyChecking=no ${SSH_CONNECTION} 'docker ps'"
                }
            }
        }
    }
}
```

7. OpenVidu 배포

- OpenVidu 설치

sudo su

cd /opt

curl https://s3-eu-west-1.amazonaws.com/aws.openvidu.io/install_openvidu_latest.sh | bash

exit

- OpenVidu 설정 및 SSL 인증서 발급 설정

sudo vi /opt/openvidu/.env

```
# 아래 부분 수정
DOMAIN_OR_PUBLIC_IP=[도메인 또는 public IP]
OPENVIDU_SECRET=[시크릿 키]
CERTIFICATE_TYPE=letsencrypt
LETSENCRYPT_EMAIL=[이메일 주소]
HTTP_PORT=8442
HTTPS_PORT=8443
```

- OpenVidu 포트 개방

```
sudo ufw allow OpenSSH
sudo ufw enable
sudo ufw allow 80/tcp
sudo ufw allow 443/tcp
sudo ufw allow 3478/tcp
sudo ufw allow 3478/udp
sudo ufw allow 40000:57000/tcp
sudo ufw allow 40000:57000/udp
sudo ufw allow 57001:65535/tcp
sudo ufw allow 57001:65535/u에
```

- OpenVidu 실행

```
# 시작
cd /opt/openvidu
sudo ./openvidu start

# 종료
cd /opt/openvidu
./openvidu stop
```

II. 외부 서비스

1. 소셜 로그인

- 구글

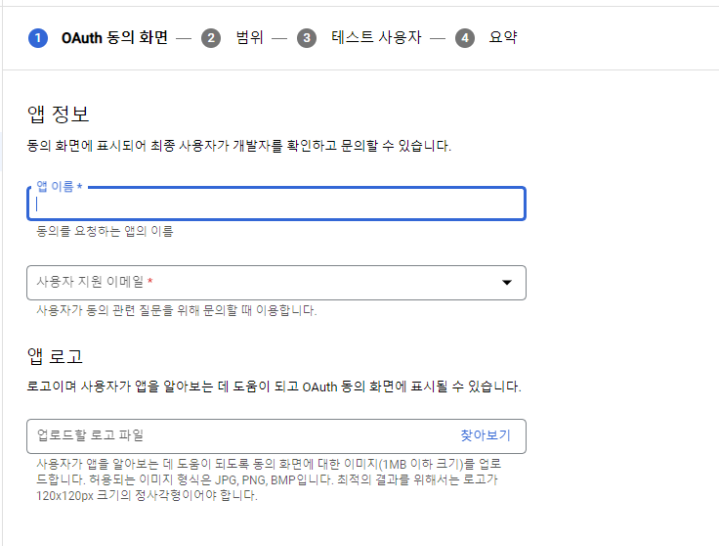
1. https://console.cloud.google.com/home/dashboard 로 접속

2. 프로젝트 선택 -> 새 프로젝트 생성 -> API 및 서비스

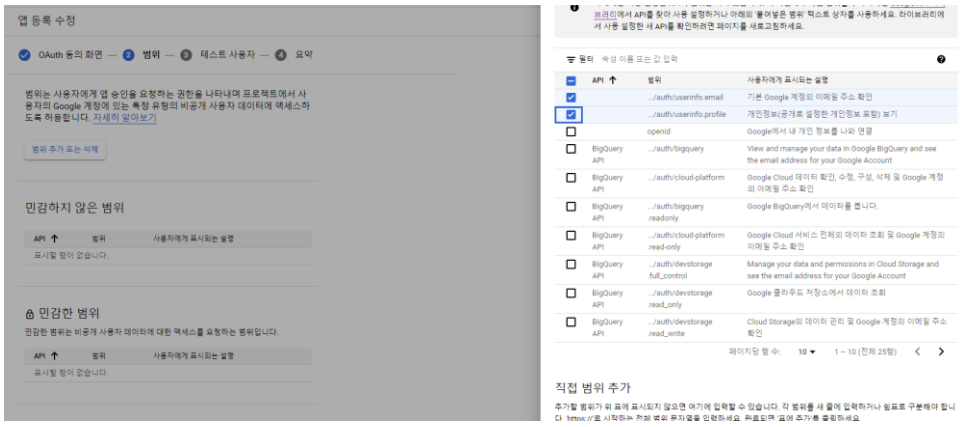
3. 사용자 인증정보 -> 사용자 인증정보 만들기 -> OAuth 클라이언트 ID -> 동의 화면 구성

4. 동의 화면에서 외부를 선택해 외부 사용자들이 사용 가능하도록 설정 -> 만들기

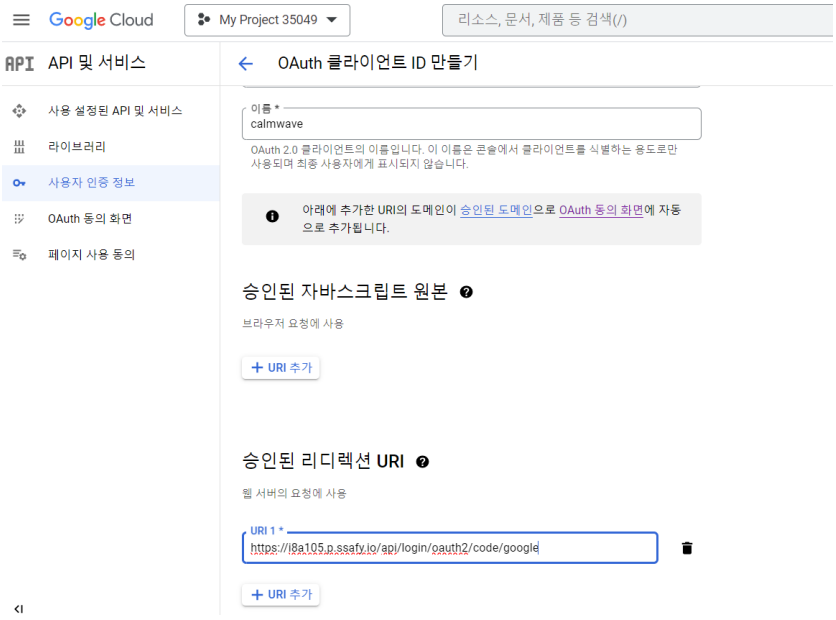
5. 앱 정보 입력.



6. 범위 추가 또는 삭제 -> email, profile 선택 후 업데이트 클릭 -> 저장 후 계속



7. 사용자 인증정보 -> 사용자 인증정보 만들기 -> OAuth 클라이언트 ID -> 웹 애플리케이션 선택 -> 승인된 리디렉션 URI 작성



2. Teachable Machine

1. https://teachablemachine.withgoogle.com/ 들어가서 포즈 프로젝트 선택

2. 클래스 생성하여 웹캠으로 촬영

3. 모델 학습시키기 버튼 클릭

4. 학습 완료 후 업로드(공유 가능한 링크) 선택 -> 모델 업로드

5. 모델 업로드가 끝나면 공유 가능한 링크와 사용할 코드 스니펫 제공해 줌. 이를 사용하여 리액트에서 개발.